D1-61

N87-24896

MEASURING SOFTWARE TECHNOLOGY

W. W. Agresti, D. N. Card, V. E. Church, and G. Page
Computer Sciences Corporation
System Sciences Division
8728 Colesville Road
Silver Spring, Maryland 20910

F. E. McGarry
National Aeronautics and Space Administration
Goddard Space Flight Center
Code 582
Greenbelt, Maryland 20771

ABSTRACT

Results are reported from a series of investigations into the effectiveness of various methods and tools used in a software production environment. The basis for the analysis is a project data base, built through extensive data collection and process instrumentation. The project profiles become an organizational memory, serving as a reference point for an active program of measurement and experimentation on software technology.

INTRODUCTION

Many proposals aimed at improving the software development process have emerged during the past several years. Such approaches as structured design, automated development tools, software metrics, resource estimation models, and special management techniques have been directed at building, maintaining, and estimating the software process and product.

Although the software development community has been presented with these new tools and methods, it is not clear which of them will prove effective in particular environments. When this question is approached from the user's perspective, the issue is to associate with each programming environment a set of enabling conditions and "win" predicates to signal when methods can be applied and which ones will improve performance. Lacking such guidelines, organizations are left to introduce new procedures with little understanding of their likely effect.

Assessing methods and tools for potential application is a central activity of the Software Engineering Laboratory (SEL) [1, 2]. The SEL was established in 1977 by the National Aeronautics and Space

Administration (NASA)/Goddard Space Flight Center (GSFC) in conjunction with Computer Sciences Corporation and the University of Maryland. The SEL's approach is to understand and measure the software development process, measure the effects of new methods through experimentation, and apply those methods and tools that offer improvement. The environment of interest supports flight dynamics applications at NASA/GSFC. This scientific software consists primarily of FORTRAN, with some assembler code, and involves interactive graphics. The average size of a project is 60,000 to 70,000 source lines of code.

SEL investigations demonstrate the advantages of building and maintaining an organizational memory on which to base a program of experimentation and evaluation. Over 40 projects, involving 1.8 million source lines of code, have been monitored since 1977. Project data have been collected from five sources:

- Activity and change forms completed by programmers and managers
- Automated computer accounting information
- Automated tools such as code analyzers
- Subjective evaluations by managers
- Personal interviews

The resulting data base contains over 25 megabytes of profile information on completed projects.

Some highlights of SEL investigations using the project history data base are presented here, organized into three sections:

- Programmer Productivity
- Cost Models
- Technology Evaluations

PROGRAMMER PRODUCTIVITY

The least understood element of the software development process is the behavior of the programmer. One SEL study examined the distribution of programmer time spent on various activities. When specific dates were used to mark the end of one phase and the beginning of the next, 22 percent of the total hours were attributed to the design phase, with 48 percent for coding, and 30 percent for testing. However, if the programmers' completed forms were used to identify actual time spent on various activities, the breakdown was approximately equal for the four categories of designing, coding, testing, and "other" (activities such as travel, training, and unknown) [3]. Although an attractive target for raising productivity was to eliminate the "other" category, the SEL found that this was not easily done.

Regarding individual programmer productivity, the SEL found differences as great as 10 to 1, where productivity was measured in lines of code per unit of effort [4]. This result was consistent with similar studies in other organizations [5].

COST MODELS

Cost is often expressed in terms of the effort required to develop software. In the effort equation,

$$E = aI^b$$

where E equals effort in staff time and I equals size in lines of code, some studies reported a value of b greater than one, indicating that effort must be increased at a higher rate than the increase in system size. The SEL analysis of projects in its data base did not support this result, finding instead a nearly linear relationship between effort and size [6]. This conclusion may be due to the SEL projects being smaller than those that would require more than a linear increase in effort.

In a separate study, the SEL used cost data from projects to evaluate the performance of various resource estimation models. One study, using a subset of completed projects, compared the predictive ability of five models: Doty, SEL, PRICE S, Tecolote, and COCOMO [7]. The objective was to determine which model best characterized the SEL environment. The results showed that some models worked well on some projects, but no model emerged as a single source on which to base a program of estimation [8]. In the SEL environment, cost models have value as a supplementary tool to flag extreme cases and to reinforce the estimates of experienced managers.

TECHNOLOGY EVALUATIONS

Several SEL experiments have been conducted to assess the effectiveness of different process technologies. One study focused on the use of an independent verification and validation (IV&V) team. The premise for introducing an IV&V team into the software development process is that any added cost will be offset by the early discovery of errors. The expected benefit is a software product of greater quality and reliability. In experimenting with an IV&V team in the SEL environment, the benefits were not completely realized [9]. The record on early error detection was better with IV&V than without it, but the reliability of the final product was not improved. Also, the productivity of the development team was comparatively low, due in part to the necessary interaction with the IV&V team. The conclusion was that an IV&V team was not effective in the SEL environment, but may be effective where there are larger projects or higher reliability requirements.

A recent SEL investigation measured the effect of seven specific techniques on productivity and reliability. From the project data base, indices were developed to capture the degree of use of quality assurance procedures, development tools, documentation, structured code, top-down development, code reading, and chief programmer team organization. The results showed that the greatest productivity and reliability improvements due to methodology use lie only in the range of 15 to 30 percent. Significant factors within this range are the positive effect of structured code on productivity and the positive effects of quality assurance, documentation, and code reading on reliability [10].

Figure 1 summarizes the perceived effectiveness of various practices in the the SEL environment [4]. The placement of the models and methods is based on the overhead cost of applying the model or method and the benefit of its use. This summary must be interpreted in the following context:

- The placement reflects subjective evaluations as well as experimental results.
- The chart is indicative of experiences in the SEL environment only.
- The dynamic nature of the situation is not apparent. The evaluation may reflect on an earlier and less effective example of the model or method.

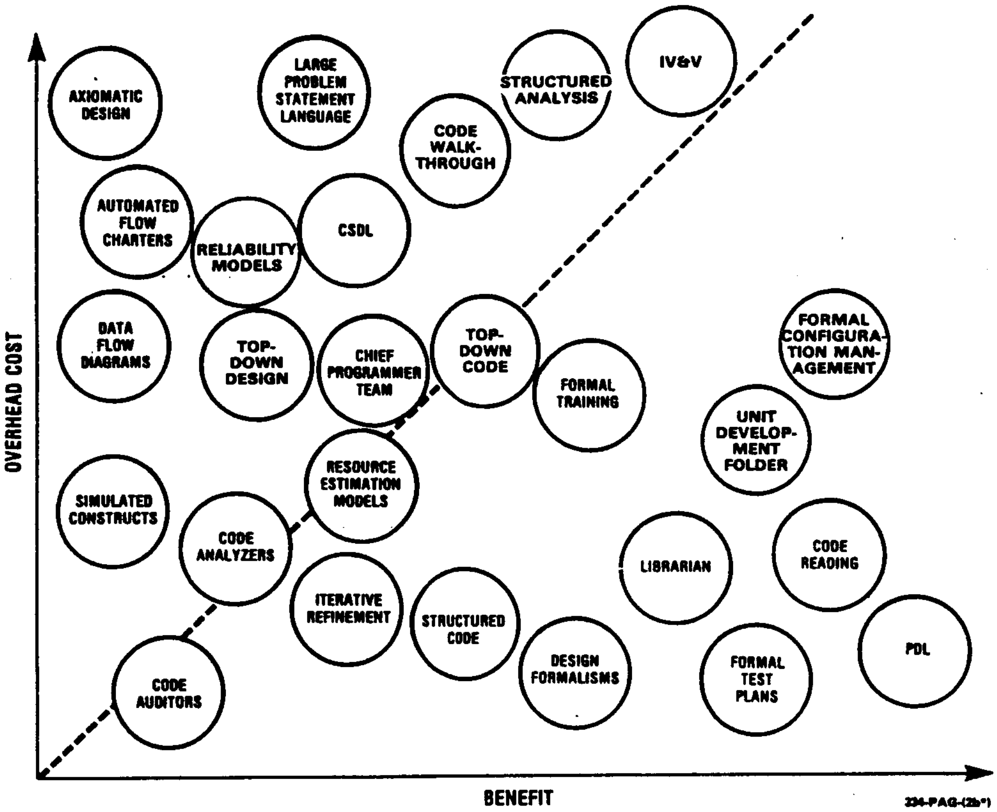

Figure 1. What Has Been Successful in Our Environment?

CONCLUSIONS

The experiences of the SEL demonstrate that statistically valid evaluation is possible in the software development environment, but only if the prerequisite quantitative characterization of the process has been obtained. Through its program of assessing and applying new methods and tools, the SEL is actively pursuing the creation of a more productive software development environment.

REFERENCES

1. V. R. Basili and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second U.S. Army/IEEE Software Life Cycle Management Workshop. New York: Computer Societies Press, 1978

2. D. N. Card, F. E. McGarry, G. Page, et al., SEL-81-104, The Software Engineering Laboratory, Software Engineering Laboratory, 1982

3. E. Chen and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

4. F. E. McGarry, "What Have We Learned in the Last Six Years Measuring Software Development Technology," SEL-82-007, Proceedings of the Seventh Annual Software Engineering Workshop, Software Engineering Laboratory, 1982

5. H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Program Performance," Communications of the ACM, January 1968, vol. 11, no.1, pp. 3-11

6. J. W. Bailey and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

7. IIT Research Institute, Quantitative Software Models, Rome Air Development Center, New York, 1979

8. J. Cook and F. E. McGarry, SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, Software Engineering Laboratory, 1980

9. G. Page, "Methodology Evaluation: Effects of Independent Verification and Integration on One Class of Application," SEL-81-013, Proceedings of the Sixth Annual Software Engineering Workshop, Software Engineering Laboratory, 1981

10. D. N. Card, F. E. McGarry, and G. Page, "Evaluating Software Engineering Methodologies in the SEL" (paper presented at Sixth Minnowbrook Workshop on Software Performance Evaluation, Minnowbrook, New York, 1983)